



Instituto de Ciência e Tecnologia
Universidade Federal de São Paulo

Compiladores: Análise Sintática LL(1)

Prof^a Thaína A. A. Tosta

tosta.thaina@unifesp.br

São José dos Campos – 2021/2

Análise Sintática Descendente Recursiva

```
void S()
{
    switch (tok)
    {
        case BEGIN : eat (BEGIN); S(); L(); break;
        case IF      : eat (IF); E(); eat (THEN); S(); eat (ELSE); S(); break;
        case PRINT  : eat (PRINT); eat(ID); break;
        default: ERRO();
    }
}
```

$S \rightarrow \text{BEGIN } S \text{ L}$ $S \rightarrow \text{IF } E \text{ THEN } S \text{ ELSE } S$ $S \rightarrow \text{PRINT ID}$
--

- Diferenciação das escolhas das regras baseada nos símbolos terminais (*tokens*);
- Nem sempre a GLC apresentará regras nesse “formato”.

Análise Sintática Descendente Recursiva

Considere a gramática G3:

$S \rightarrow E \text{ EOF}$
 $E \rightarrow E + T$
 $E \rightarrow E - T$
 $E \rightarrow T$
 $T \rightarrow T * F$
 $T \rightarrow T / F$
 $T \rightarrow F$
 $F \rightarrow \text{ID}$
 $F \rightarrow \text{NUM}$
 $F \rightarrow (E)$

```
void S()  
{  
    E(); eat (EOF); break;  
}
```

```
void E()  
{  
    switch (tok)  
    {  
        case ??? : E(); eat (PLUS); T(); break;  
        case ??? : E(); eat (MINUS); T(); break;  
        case ??? : T(); break;  
        default: ERRO();  
    }  
}
```

Não há como saber qual alternativa escolher.

Análise Sintática Descendente Recursiva

- Analisadores descendentes recursivos só podem ser construídos p/ gramáticas em que o primeiro símbolo terminal de cada regra fornece informações suficientes para escolher a produção a ser utilizada;
- Problemas nessa abordagem:
 - $A \rightarrow \alpha \mid \beta$: α e β iniciarem com símbolos não-terminais
 - $A \rightarrow \epsilon$: O não-terminal A pode desaparecer, e precisamos saber quais *tokens* podem suceder A legalmente.

Análise Sintática LL(1)

- Uma solução para esses problemas é o uso da gramática LL(1):
 - Primeiro L: a cadeia de entrada (sentença de *tokens*) é obtida da esquerda para a direita (*left to right*);
 - Segundo L: adota-se derivação mais à esquerda (*leftmost derivation*);
 - Número (1): indica o número de símbolos de entrada (*tokens*) necessários p/ decidir qual produção será escolhida (*lookahead*).
- A gramática LL(1) leva à implementação do *parser* LL(1).

Análise Sintática LL(1)

- *O parser* LL(1) utiliza uma pilha explícita em vez de ativações recursivas;
- Portanto, implementa uma análise sintática descendente, não mais recursiva, chamada análise sintática LL(1).

Análise Sintática LL(1)

Modelo básico da análise sintática LL(1)

- Considere a GLC: $S \rightarrow (S) S \mid \epsilon$ que gera cadeias de parênteses balanceados;

	Pilha de Análise Sintática	Entrada	Ação
1	\$ S	() \$	$S \rightarrow (S) S$ [substitui]
2	\$ S) S (() \$	[casamento]
3	\$ S) S) \$	$S \rightarrow \epsilon$ [substitui]
4	\$ S)) \$	[casamento]
5	\$ S	\$	$S \rightarrow \epsilon$ [substitui]
6	\$	\$	[aceita]

- Quando a pilha e a entrada estiverem vazias, então a análise chegou ao fim, sem identificação de erros.

Análise Sintática LL(1)

Técnica de implementação do *parser* para gramática LL(1)

- Cria-se uma tabela preditiva $M[N,T]$ para auxiliar na construção do *parser*;
- N é o conjunto de símbolos não-terminais da gramática;
- T é o conjunto de símbolos terminais da gramática;
- Suponha que tenhamos o não-terminal A e o terminal a :
 - Na posição $A \times a$ da tabela devemos colocar qual é a produção que devemos utilizar se, ao tentarmos reconhecer A , encontramos na entrada o símbolo a .

Análise Sintática LL(1)

Exemplo de tabela preditiva

$S \rightarrow aSAb$

$S \rightarrow bAa$

$A \rightarrow bAb$

$A \rightarrow c$

Não-Terminal	Terminal		
	a	b	c
S	$S \rightarrow aSAb$	$S \rightarrow bAa$	
A		$A \rightarrow bAb$	$A \rightarrow c$

- As posições vazias na tabela indicam que esse terminal não pode aparecer no início da regra de produção;
- Se tivermos mais do que uma produção em alguma posição da tabela, então a GLC não pode ser usada na construção de um *parser* LL(1).
 - A gramática não é do tipo LL(1).

Análise Sintática LL(1)

- Para que o *parser* LL(1) seja implementado com sucesso, é necessário garantir:
 - Que as regras de produção da GLC não apresentem recursão à esquerda por uma **técnica da remoção de recursão à esquerda**;
 - Que a disposição dos símbolos terminais nas regras da GLC permitam a escolha de uma única produção pela **técnica de fatoração à esquerda**.

Análise Sintática LL(1)

Recursão à Esquerda

- Ocorre quando, p/ algum não-terminal B , temos:
 $B \rightarrow B\alpha$ (α significa uma cadeia de terminais ou não-terminais);
- Intuitivamente é fácil perceber que o *parser* poderia entrar em uma recursão infinita, ao tentar reconhecer B sem consumir nenhum *token* de entrada.

Análise Sintática LL(1)

Recursão à Esquerda

- Pode ser de dois tipos:

- Direta (ou imediata)

$$B \rightarrow B\beta$$

- Indireta

$$B \rightarrow A\alpha$$

$$A \rightarrow B\beta$$

Análise Sintática LL(1)

Remoção da recursão à Esquerda direta (reescrevendo a gramática em notação BNF)

- 1) Dividir as produções de B em dois subconjuntos:
 - $N = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$, das produções que não possuem recursão à esquerda;
 - $R = \{B\beta_1, B\beta_2, \dots, B\beta_m\}$, das produções que possuem recursão à esquerda.
- 2) Eliminar as produções de R da gramática.

Análise Sintática LL(1)

Remoção da recursão à Esquerda direta

3) Adicionar à gramática as seguintes produções:

$B \rightarrow \alpha_1 B'$ (α_i representa a parte sentencial das regras sem recursão à esquerda)

$B \rightarrow \alpha_2 B'$

.....

$B \rightarrow \alpha_n B'$

$N = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$, das produções que não possuem recursão à esquerda;

$R = \{B\beta_1, B\beta_2, \dots, B\beta_m\}$, das produções que possuem recursão à esquerda.

onde B' é um novo símbolo não-terminal.

Análise Sintática LL(1)

Remoção da recursão à Esquerda direta

4) Adicionar à gramática as novas produções p/ B'

$$B' \rightarrow \beta_1$$

$$B' \rightarrow \beta_2$$

.....

$$B' \rightarrow \beta_m$$

β_i representa a parte sentencial das regras com recursão à esquerda, após o símbolo não-terminal recursivo

$$B' \rightarrow \beta_1 B'$$

$$B' \rightarrow \beta_2 B'$$

.....

$$B' \rightarrow \beta_m B'$$

$N = \{\alpha_1, \alpha_2, .. \alpha_n\}$, das produções que não possuem recursão à esquerda;

$R = \{B\beta_1, B\beta_2, .. B\beta_m\}$, das produções que possuem recursão à esquerda.

Análise Sintática LL(1)

Exemplo: Utilizar o procedimento descrito p/ eliminar a recursão à esquerda nas produções da seguinte gramática:

$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow \text{NUM}$$

- α β
- ↓ ↓
- 1) $N = \{E \rightarrow T\}$, $R = \{E \rightarrow E + T\}$
- 2) $R = \{E \rightarrow E + T\}$
- 3) $E \rightarrow T E'$
- 4) $E' \rightarrow +T \mid +T E'$

Gramática modificada

$$E \rightarrow TE' \mid T$$

$$E' \rightarrow +TE' \mid +T$$

$$T \rightarrow \text{NUM}$$

Análise Sintática LL(1)

Gramática original

$$\begin{aligned} E &\rightarrow E + T \\ E &\rightarrow T \\ T &\rightarrow \text{NUM} \end{aligned}$$

Gramática modificada

$$\begin{aligned} E &\rightarrow TE' \mid T \\ E' &\rightarrow +TE' \mid +T \\ T &\rightarrow \text{NUM} \end{aligned}$$

5 + 8 + 2

5 + 8 + 2

$E \Rightarrow E + T$

$\Rightarrow E + T + T$

$\Rightarrow T + T + T$

...

$\Rightarrow \text{NUM} + \text{NUM} + \text{NUM}$ OK

5 + 8 + 2

$E \Rightarrow TE'$

$\Rightarrow \text{NUM } E'$

$\Rightarrow \text{NUM} + TE'$

$\Rightarrow \text{NUM} + \text{NUM } E'$

$\Rightarrow \text{NUM} + \text{NUM} + T$

$\Rightarrow \text{NUM} + \text{NUM} + \text{NUM}$ OK

Análise Sintática LL(1)

Exercício: Utilizar o procedimento descrito p/ eliminar a recursão à esquerda nas produções da gramática:

$\text{expressão} \rightarrow \text{expressão} + \text{termo}$

$\text{expressão} \rightarrow \text{expressão} - \text{termo}$

$\text{expressão} \rightarrow \text{termo}$

$\text{termo} \rightarrow \text{termo} * \text{fator}$

$\text{termo} \rightarrow \text{termo} / \text{fator}$

$\text{termo} \rightarrow \text{fator}$

$\text{fator} \rightarrow (\text{expressão})$

$\text{fator} \rightarrow \text{ID}$

$\text{fator} \rightarrow \text{NUM}$

Gramática modificada

$\text{expressão} \rightarrow \text{termo expressão}' \mid \text{termo}$

$\text{expressão}' \rightarrow -\text{termo} \mid +\text{termo} \mid -\text{termo expressão}' \mid +\text{termo expressão}'$

$\text{termo} \rightarrow \text{fator termo}' \mid \text{fator}$

$\text{termo}' \rightarrow * \text{fator} \mid / \text{fator} \mid * \text{fator termo}' \mid / \text{fator termo}'$

$\text{fator} \rightarrow (\text{expressão}) \mid \text{ID} \mid \text{NUM}$

Análise Sintática LL(1)

Fatoração à Esquerda

Considere a seguinte gramática

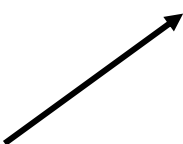
$S \rightarrow \text{if } E \text{ then } S \text{ else } S$

$S \rightarrow \text{if } E \text{ then } S$

...

Tentativa de construir uma função p/ reconhecer a produção

S: void S()
 {
 case if: eat (if); E(); eat (then); S(); eat (else); S(); break;
 case if: eat (if); E(); eat (then); S(); break;
 }



Problema: o mesmo terminal “if” inicia a produção S. Logo, não há como escolher a produção correta, indicando que essa não é uma gramática LL(1).

Análise Sintática LL(1)

Fatoração à Esquerda

- Esse problema pode ser resolvido utilizando-se o procedimento chamado **FATORAÇÃO À ESQUERDA**;
- A fatoração à esquerda consiste em modificar as produções de um não-terminal de modo a adiar a decisão sobre qual produção utilizar, até que tenham sido lidos *tokens* suficiente p/ isso;
- Basicamente, deve ser identificado o **maior prefixo comum** das produções a serem fatoradas, e criar novas produções p/ completar a produção original.

Análise Sintática LL(1)

Fatoração à Esquerda

$S \rightarrow \text{if } E \text{ then } S \text{ else } S$

$S \rightarrow \text{if } E \text{ then } S$

A produção S acima, após a fatoração à esquerda ficaria assim:

$S \rightarrow \text{if } E \text{ then } S X$ *prefixo comum: if E then S*

$X \rightarrow \text{else } S$

$X \rightarrow \epsilon$

*X: parte variável, pode ser else S, ou vazio
X é um não-terminal anulável*

Análise Sintática LL(1)

Fatoração à Esquerda

$S \rightarrow \text{if } E \text{ then } S X$

$S \rightarrow \text{print id}$

$X \rightarrow \text{else } S$

$X \rightarrow \varepsilon$

```
void S()
{
    switch(tok)
    {
        case if: eat (if); E(); eat (then); S(); X(); break;
        case print: eat(print); eat(id);
        default: ERRO();
    }
}

void X()
{
    switch(tok)
    {
        case else: eat(else); S(); break;
    }
}
```

Análise Sintática LL(1)

Aplicar o procedimento de fatoração à esquerda para a gramática a seguir:

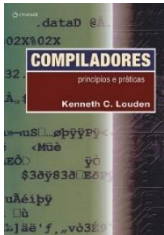
expressão \rightarrow termo expressão' | termo
expressão' \rightarrow -termo | +termo | -termo expressão' | +termo expressão'
termo \rightarrow fator termo' | fator
termo' \rightarrow *fator | /fator | *fator termo' | /fator termo'
fator \rightarrow (expressão) | ID | NUM

Gramática modificada

expressão \rightarrow termo expressão'
expressão' \rightarrow -termo expressão' | +termo expressão' | ε
termo \rightarrow fator termo'
termo' \rightarrow *fator termo' | /fator termo' | ε
fator \rightarrow (expressão) | ID | NUM

Análise Sintática LL(1)

Bibliografia consultada



LOUDEN, K. C. **Compiladores: princípios e práticas.** São Paulo: Pioneira Thompson Learning, 2004.

MERINO, M. **Notas de Aulas - Compiladores,** UNIMEP, 2006.